Laboratorul 06.

Arhiva laborator 6: arhiva6.zip

Problema 0 - vezi curs!!

Să se definească o clasă SortedVector derivată din Vector, care să permită ordonarea după orice criteriu, specificat de utilizator la construirea unui obiect SortedVector. Clasa va conţine o variabilă de tip Comparator, iniţializată de un constructor cu argument de tip Comparator şi folosită de metoda Collections.sort.

Observatie!! Vom supradefini doar **add**-ul din Vector pt aceasta problema, pt a fi mai simplu. In mod normal ar trebui supradefinite toate metodele de adaugare, setare etc care ne-ar putea modifica ordinea!

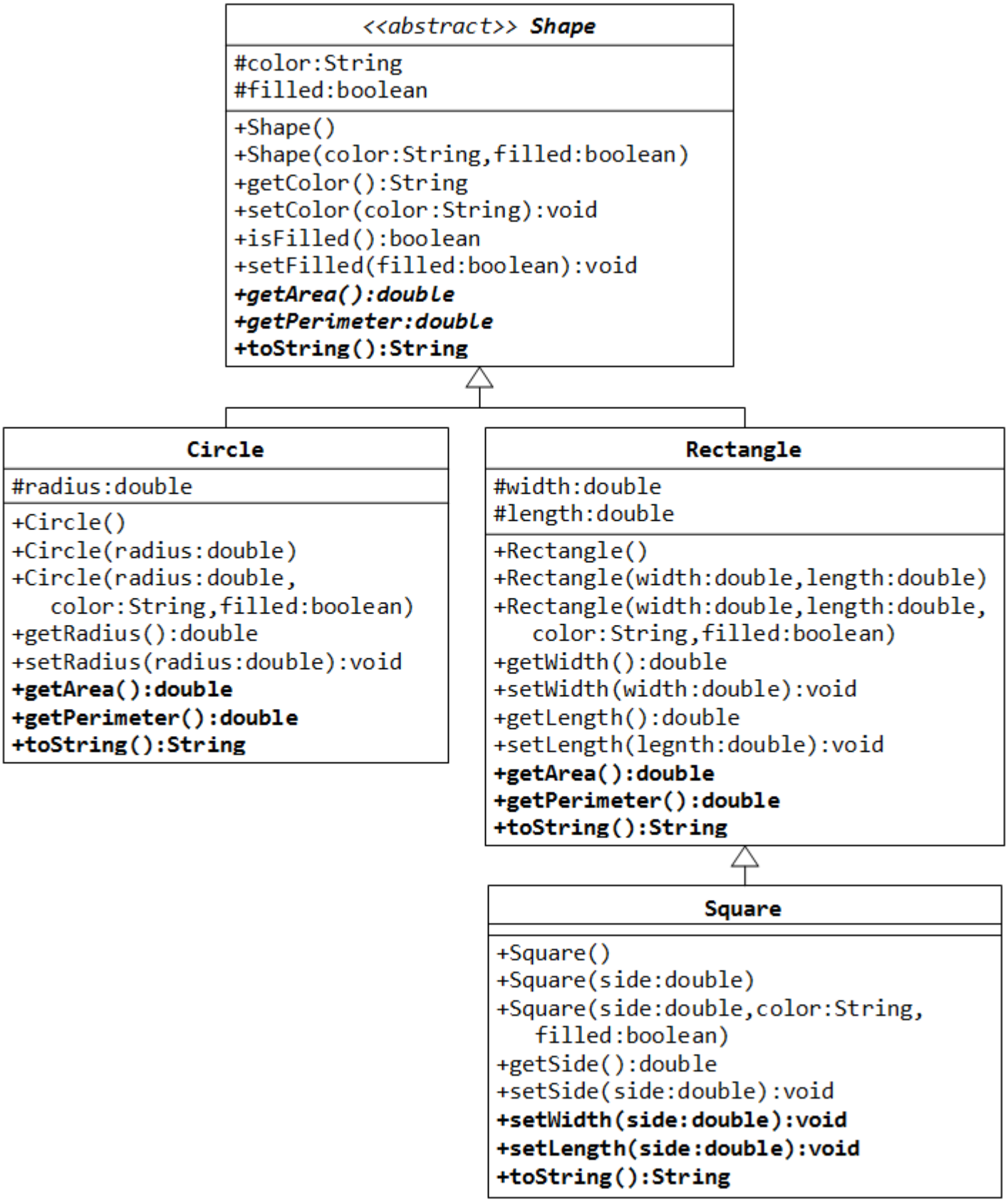
Să se definească o clasă Pair care conţine două date de tip Object, cu metodele equals şi toString redefinite.

Să se scrie două clase Comparator pentru compararea obiectelor Pair dupa primul obiect, respectiv, comparare după al doilea obiect din pereche

Să se scrie un program pentru crearea a doi vectori SortedVector de obiecte Pair, unul ordonat după primul obiect din pereche şi celălalt ordonat după al doilea obiect: vom adăuga obiecte Pair ce conţin un String şi un Integer.

Problema 1

Să se implementeze ierarhia de clase descrisă prin diagrama de mai jos, astfel incat constructorii subclaselor sa foloseasca cat mai mult constructorii superclaselor cat si cei proprii (*super(..)*, *this(..)*):



Clasa abstractă ***Shape*** conține trei metode abstracte: ***getArea***, ***getPerimeter*** și metoda ***toString*** care este implementată în fiecare clasă și este particularizată, pentru a face distincția între obiecte. Aveti aceasta clasa in arhiva laboratorului!!

Pentru validarea acestei cerințe, puteți folosi clasa ***Test01*** pusă la dispoziție în arhiva laboratorului.

Problema 2

Pornind de la ierarhia definită anterior, declarati un obiect de tip ***Rectangle***. Verificați dacă puteți instanția un astfel de obiect folosind unul din constructorii clasei ***Shape*** și unul din cei existenți în clasa ***Square***. Declarați și instanțiați un obiect de tip ***Square***, apoi incercați să îi faceți cast la un obiect de tip: ***Rectangle***, ***Shape*** și ***Circle***. Eliminați erorile apărute, folosind ***instanceof*** pentru a determina dacă este validă operația pe care incercați să o realizați.

Determinați tipurile conversiilor, din blocul de cod de mai jos, indicând dacă sunt sau nu posibile.

```
//Conversie 1
Circle c1 = new Circle();
Square sq = (Square) c1;
//Conversie 2
Rectangle r = new Rectangle(5.0, 5.0);
sq = (Square) r;
//Conversie 3
sq = new Square(7.0);
r = sq;
```

Upcasting & **Downcasting**

Problema 3

Pornind de la interfața propusă, creați o clasă ***Student*** care implementează interfața ***Persoana*** și conține ca membri privați numele studentului și un vector cu elemente de tip Double, reprezentând mediile acestuia. Compararea se va realiza crescator dupa nume, iar dacă există doi studenți care au același nume, primul este cel cu media mai mare.

Scrieți o metodă ***main*** care construiește un vector (obiect din clasa ***Vector***) cu elemente de tip ***Student***, îl ordonează, conform criteriului anterior, folosind Collectios.sort, și apoi îl afișează!

```
interface Persoana extends Comparable {
    public double calculMedieGenerala();
    public String getNume();
    public void setNume(String nume);
    public void addMedie(double medie);
}
```

```
Collections.sort
```

Problema 4

Să se definească o clasă filtru pentru selecția de fișiere după o listă de extensii (tipuri de fișiere) și după o listă de nume. Clasa implementează una din interfețele ***FileFilter*** sau ***FilenameFilter*** conține un vector cu tipurile de fișiere acceptate și un vector cu cuvintele care ar trebui să între în alcătuirea numelor fișierelor selectate.

Pentru testare, se poate folosi ierarhia de directoare pusă la dispoziție în arhiva laboratorului. Extensiile acceptate se citesc din fișierul **extension.in**, iar cuvintele din fișierul **words.in**, folosind obiecte de tipul ***RandomAccessFile***. Fisierele si ierarhia pe care puteti lucra se regasesc in arhiva laboratorului.

Problema 5

Să se modeleze o garnitură de tren. Se va defini în acest scop o clasă ***Tren***. Un obiect de tip ***Tren*** conține mai multe referințe spre obiecte de tip ***Vagon*** care sunt păstrate într-un obiect de tip ***Vector***. Un vagon poate fi de 3 tipuri: ***CalatoriA***, ***CalatoriB*** și ***Marfa***. Despre garnitura de tren și vagoane mai cunoaștem următoarele:

- un tren poate conține mai multe tipuri de vagoane;
- un vagon de tip ***CalatoriA***:
    1. are capacitatea de 40 pasageri și 300 colete poștale;
    2. furnizează două servicii pentru deschiderea, respectiv închiderea automată a ușilor.
- un vagon de tip ***CalatoriB***:
    1. are capacitatea de 50 pasageri și 400 colete poștale;
    2. furnizează două servicii pentru deschiderea, respectiv închiderea automată a ușilor;
    3. fiind un vagon mai nou, furnizează un serviciu automat pentru blocarea geamurilor.
- un vagon de tip ***Marfa***:
    1. are capacitatea de 400 colete poștale;
    2. nu furnizează servicii pentru deschiderea, respectiv închiderea automată a ușilor, aceste operații executându-se manual.

Implementați clasa **abstractă** care modelează conceptul de vagon împreună cu clasele sale derivate. Se menționează că apelurile serviciilor pentru deschiderea, respectiv închiderea ușilor și blocarea geamurilor vor afișa pe ecran un mesaj corespunzător.

Implementați clasa ***Tren*** conform specificațiilor. Obiectele de tip ***Tren*** trebuie să fie comparabile, două trenuri fiind considerate egale dacă pot transporta același număr de colete. Această clasă conține o metodă pentru afișarea tipurilor de vagoane existente în componența trenului.

Pentru testare, se poate **completa** clasa ***Test05*** pusă la dispoziție în arhiva laboratorului.

poo/laboratoare/06.txt · Last modified: 2021/11/14 20:53 by carmen.odubasteanu